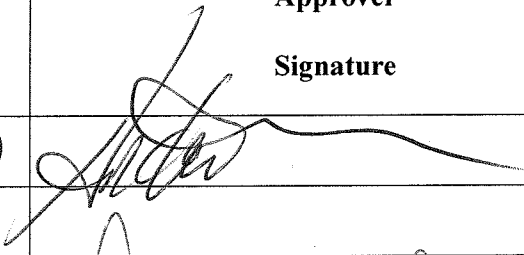
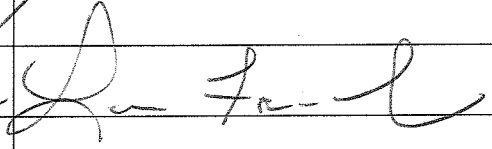


<p align="center">BEACON HR/PAYROLL IMPLEMENTATION PROJECT DEVELOPMENT STANDARDS</p>

DOCUMENT HISTORY

Date	Document Revision	Document Revision Description	Document Author
04/17/06	1.0	Initial Release	Richard Fox
04/24/06	1.1	Revised by Martin Geres	
05/22/06	1.2	Added new topics	Anjani Vemula

APPROVERS

Approval Date	Approver Name and Role	Approver Signature
7/13/06	ANDREW KOENIGSBERG, Deputy PD	
7-31-06	Don Foehr, PGM mgr	

PMO RELEASE AUTHORIZATION

Release Authority: Mya Powell Date: 8-1-06
Version#: 1.0

TABLE OF CONTENTS

1. INTRODUCTION	1
2. SAP NAMING CONVENTIONS.....	1
2.1. SAP Object Naming Conventions	1
2.2. Naming Conventions for Programs, Functions, Includes, Interfaces and Reports	2
2.3. Naming conventions for function groups	2
2.4. Naming convention for Function Modules.....	3
2.5. Naming Conventions for Transaction Codes.....	3
2.6. Naming Conventions for Packages.....	3
2.7. Naming Conventions for Data Dictionary.....	4
2.8. Other naming conventions.....	7
3. ABAP/4 STANDARD CONVENTIONS	10
3.1. Program Attributes	10
3.2. Internal Table Naming Convention	11
3.3. System Table Addressing Convention	11
3.4. Program Variable Names.....	12
3.5. Error Messages	15
3.6. PF-Status.....	15
3.7. External Files.....	16
3.8. Variants.....	16
4. ABAP/4 PROGRAMMING STANDARDS AND PRACTICES.....	16
4.2. ABAP Program Layout	17
4.3. Program Structure.....	18
4.4. Main Documentation Block.....	18
4.5. Statement Formatting	19
4.6. Include Code.....	19
4.7. Messages.....	20
4.8. Report Formatting	20
4.9. Batch Data Input.....	21
4.10. Data Access	21
4.11. Data Update	21
4.12. External File Usage	21
4.13. Variants.....	21
4.14. Computations.....	22
4.15. Subroutines	22
4.16. Authorization Checking.....	22
4.17. Lock objects.....	22
4.18. Debugging	22
4.19. Standards for Specific Statements	22

4.20.	User Interface (GUI).....	25
4.21.	Maintenance/Correction of existing production code.....	25
4.22.	External Data Sets	25
4.23.	Internal Tables	26
5.	PROGRAM LOGIC	27
5.1.	STANDARD	27
5.2	GUIDELINES	28
6.	DATA ACCESS	28
6.1	DATA ACCESS – STARNDARD RULES.....	28
6.2.	DATA ACCESS – GUIDELINES	29
6.3	DATA ACCESS – TIPS	30
7	ABAP/4 MODULE POOL STANDARDS AND PRACTICES.....	30
7.1	Program Structure.....	31
7.2	Screen Definitions	31
7.3	GUI interfaces	31
7.4	POP-UP windows.....	31
8	REFERENCES.....	31
9	31	
UNIT TESTING:	31
10	APPENDIX A.....	32
10.1	Table of Business Object Codes.....	32
11	APPENDIX B	32
11.1	Suffix Codes for Field Names	32
12	APPENDIX C	33
13	APPENDIX D	33
13.1	ABAP Check list	33

1. INTRODUCTION

This paper is intended for consultants and internal developers to help define a strategy and implementation specific development guidelines. This paper should also be used in conjunction with any SAP provided tools or templates for this topic. It is recommended that the reader of this document have been exposed to the some basic concepts within the SAP product and had previous development experience. Attending SAP training course for SAP R/3 Overview, ABAP Workbench and/or an equivalent is recommended.

2. SAP NAMING CONVENTIONS

2.1. SAP Object Naming Conventions

The purpose of this section is to define the naming conventions for the following object types found in the SAP environment:

- Programs, Functions, Includes, and Reports
- Module Pools, Module Pool Includes
- Function Groups, Modules
- Transaction Codes
- Packages
- Logical Databases
- SAP Standard objects
- Data Dictionary elements.

As a result of RICE workshops, all custom development will be identified and assigned a Development ID. This ID will consist of a character identifying the type of development (R = Report, I = Interface, C = Conversion, S = Sapscrip/smartform, A = Custom development, U = Utility type programs, M = Module Pools, L = Include programs) and a sequential three-digit number. The Development ID will be a significant component used in the naming of related development objects.

EX: A009 – Custom dunning program

2.2. Naming Conventions for Programs, Functions, Includes, Interfaces and Reports

The purpose of this section is to define the naming conventions to be used by ABAP/4 programs, functions, includes, interfaces, and reports created in the SAP system. Object names will be formatted as follows:

Position

1	Usage: "Y" Tools and Utilities that are not to be migrated to production. "Z" Application programs that will be moved to production.
2-5	Development ID
6-7	SAP Business Object area (Ref. Appendix A)
8	"_" Underscore
9-20	Related or meaningful 15 character acronym or paraphrase; should tie to the application's function

Use the underscore character "_" as a spacer or word separator when needed.

The following are example program names for the R/3 Business Object Area of General Ledger:

- Interface name ZI001GL_MONTHLY_POST
- Conversion name ZC003MM_LOAD_MATERIAL
- Report Name ZR003GL_ACCOUNT_SPLIT
- Include name ZA009SD_CUSTOMER_TOP.

2.3. Naming conventions for function groups

The purpose of this section is to define the naming conventions to be used for function groups. Function group names formatted as follows:

Position

1	"Z".
2-5	Development ID
6-7	SAP Business Object area (Ref. Appendix A)
8	"G" for function Group
9	(Underscore)
10-25	Function Group Name (words separated by underscores)
26-28	Not used

- Function groups should contain related function modules and have the global interface flag turned off
- Ex. ZI01SDG_CONTRACTS.

2.4. Naming convention for Function Modules

The purpose of this section is to define the naming conventions to be used for function modules. Function module names are recommended to be very descriptive. The following is a suggested naming convention:

Position

1	"Z" required by SAP
2-5	Development ID
6-7	SAP Business Object area (Ref. Appendix A)
8	"F" for Function Module
9	(Underscore)
10-31	Use descriptive words, separated by underscores

- Function modules should always reference data dictionary structures in the interfaces for export, import and tables. This is mandatory if the function module being created is to be used for RFC interfaces
- Ex. ZI001SDF_GET_CONTRACT.

2.5. Naming Conventions for Transaction Codes

The purpose of this section is to define the naming conventions to be used for transaction codes (TCODE) created in the SAP system. Transaction code names will consist of up to (20) characters formatted as follows:

Position

1	"Z"
2-5	Development ID
6-20	Descriptive Name (if required)

- Ex. ZI001
- Ex. ZI001RUNONCE

2.6. Naming Conventions for Packages.

The purpose of this section is to define the naming conventions to be used for packages (stored in the TDEV table). Packages are used to pool related SAP objects into one common area for easier reference. Development class names will be formatted as follows:

Position

1	"Z".
2-3	SAP Business Object area (Ref. Appendix A)
4-6	<p>'Ann' where nn is a number between 00 and 99 and A is for Functional configurations or the first character of the type of the development as listed below:</p> <p>R = Reports I = Interfaces C = Conversions S = Sapscripts/smartforms A = Custom U = Utility type programs M = Module Pools L = Include programs</p>

- Ex. ZSDR001, ZBCR002

2.7. Naming Conventions for Data Dictionary

2.7.1. Tables, Views and Structures

Custom tables are tables maintained exclusively by custom on-line programs. It is recommended that all tables be defined as transparent tables. Tables can also be maintained via the maintain tables transaction SM31.

The same holds true for Views and Structures

Position

1	"Z", required by SAP
2-16	User defined
	* Append _V for views

- Ex: ZDEALERS – Dealer master table/structure

2.7.2. Views (See Tables)

2.7.3. Structures (See Tables)

2.7.4. Domains

Domains define the physical attributes of a field. Domains can be checked against value tables or be assigned a check table depending on the usage for the domain. Domain names can be 30 characters long,

but should be kept as short as possible, and are to be defined by the following naming standards by position:

Position

1	"Z" Required by SAP
2-30	User Defined

- Ex: ZCONTR – Contract Number

2.7.5. Data Elements

Data elements define the description for a field (Semantic domain). Data elements are to reference a corresponding domain that matches the physical requirements for that element. Data element names can be 30 characters long and are defined by the following naming standards by position:

Position

1	"Z" Required by SAP
2-30	User Defined

- Ex: ZCONTR – Contract Number

Documentation must be maintained for the data element. This documentation appears as help text when fields in an on-line program are tied to the data element.

2.7.6. Fields

Fields are to be descriptive. Long field names are cumbersome from a programming perspective. It is recommended that the field name be named according to the associated data element in the following convention.

Position

1-n	Data element name, drop the Z for custom data elements.
-----	---

- Ex: CONTR – Contract Number
- *Note: Fields of Append Structures MUST begin with ZZ (This ensures future SAP field names within the table will not be named identically). Please explain why the z is dropped for custom data elements? Explanation to drop Z for a field name:- In a Z table, we could define the fields by dropping Z and define the fields of Z table as the standard data fields referencing standard data element, so that we will get the standard characteristics of that field such as Data, Length, Decimals and Text and gets populated automatically. Also, this will allow using commands such as move-corresponding etc.

SAP Dictionary: Display Table

Transp. Table: PA2001 Active
Short Text: HR Time Record: Infotype 2001 (Absences)

Attributes | Delivery and Maintenance | **Fields** | Entry help/check | Currency/Quantity Fields

Field	Key	Initi...	Data element	Data T...	Length	Deci...	Short Text
MANDT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDT	CLNT	3	0	Client
INCLUDE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	PAKEY	STRU	0	0	Key for HR Master Data
PERNR	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	PERSNO	NUMC	8	0	Personnel number
SUBTY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	SUBTY	CHAR	4	0	Subtype

1 / 88

E18 (2) (100) ncsperssb801 INS

SAP Dictionary: Display Table

Transp. Table: ZHR_ABS_DEMO Active
Short Text: Annual Absences for Demo Uploads

Attributes | Delivery and Maintenance | **Fields** | Entry help/check | Currency/Quantity Fields

Field	Key	Initi...	Data element	Data T...	Length	Deci...	Short Text
MANDT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDT	CLNT	3	0	Client
ORG UNIT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	HROBJID	NUMC	8	0	Object ID
PERNR	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	PERSNO	NUMC	8	0	Personnel number
WB1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZABSTY	CHAR	2	0	Absence Type Selection

1 / 57

E18 (1) (100) ncsperssb801 INS

Please see the above screen shots. We have standard SAP Table PA2001, which has fields such as MANDT, PERNR etc. In the ZHR_ABS_DEMO table, we have MANDT and PERNR which are referring to Data elements MANDT and PERSNO.

In the table under 2.8 sap is recommending to start with z or y ?? – Explanation: We could start with Z or Y. In most our clients we start with Z for all the custom developed objects that will be transported to production and where as all the objects starting with Y would be local objects and would not be attached to any transport. Project Tech Team leadership will take necessary decision whether to start with Z or Y.

2.8. Other naming conventions

What is the purpose of listing them? Do we adhere to them if not already specified otherwise? These are the general guidelines for naming conventions and the developer will look for these guidelines whenever developing a particular object.

A list of other naming conventions suggested by SAP (from 4.6c help) has been listed below:

Object	Length	Customer Name range
Application log	4	Y* OR Z*
Object	10	Y* OR Z*
Subobject		
Authorization/authorization profile	12	No " " in second position
Authorization object	10	Y* OR Z*
Authorization object class	4	Y* OR Z*
Authorization object group	30	Y* OR Z*
CATT procedure	30	Y* OR Z*
Change document object	10	Y* OR Z*
Class	30	Y* OR Z* (Underscore permitted)
Code page	4	9000-9999
Data element	30	Y* OR Z*
Development class	30	Refer to previous sections
Dialog module	30	Y* OR Z* RP_9* RH_INFOTYP_P9* (include Development ID in the 2nd and the 3rd character)

Documentation module	10	Like object name
Authorization object	12	Like object name
Authorization profile	20	Y* OR Z*
Chapter in a structure (CHAP)	20	Y* OR Z*
Description of a CATT procedure	28	Y* OR Z*
Dialog text	28	Y* OR Z*
General text (TX)	20	Y* OR Z*
Implementation Guide chapter	20	Y* OR Z*
Main chapter in a structure (BOOK)	26	Y* OR Z*
Note on chapter in a structure (NOTE)	20	Y* OR Z*
Release Notes	12	
Structure		
Domain	30	Y* OR Z*
Enhancement	8	Y* OR Z*
Enhancement project	8	*
Enterprise Data Model (EDM)	10	Y* OR Z*
Data model	10	Y* OR Z*
Entity	10	Y* OR Z*
Function Builder	30	Y * Z *
Function module	26	Refer Previous section
Function group	4	Refer previous section
User exit function module	4	Similar to Function module
User exit function module (customer-specific)		
IDoc development	27	Z1* (include development ID)
Segment type	30	Z2*
Segment name	30	Y* OR Z*
Basic IDoc type	30	*
Enhancement type	30	*
Logical message		
Info type number	4	9000 – 9999
Interface	30	Y* OR Z* (Underscore permitted)
LIS (Logistics Information System)	2	Y* OR Z*
Event	2	Y* OR Z*
Unit		
Lock object	16	EY* EZ*
Logical database	20	Y* OR Z*
Maintenance and transport object	31	Y* OR Z*
Matchcode	1	0 – 9
Matchcode ID	4	Y* OR Z*
Matchcode object		
Menu	20	Y* OR Z*

Message	20	Y* OR Z*
Message ID	3	900 - 999
Message number		
Module pool	30	SAPDY* SAPDZ*
Module pool for dialog	30	DY* DZ*
INCLUDES	8/40	SAPMY* SAPMZ*
Module pool for screens	30	MY* MZ*
INCLUDES	30	MP9*
Module pool for info types	30	MP9*
INCLUDES	30	SAPFY* SAPFZ*
Module pool for subroutines	30	FY* FZ*
INCLUDES	30	SAPUY* SAPUZ*
Module pool for update program	30	UY* UZ*
INCLUDES		
Number range document object	10	Y* OR Z*
Pool name/cluster name	10	Y* OR Z*
Printer macro	-	Y* OR Z* or 9*
R/3 Analyzer: Identifier	20	Y* OR Z*
Relation ID	2	Y* OR Z*
Report	30	Refer to previous sections
Report category	4	Y* OR Z*
Report variant	14	X* OR CUS&*
Transportable, global	14	Y*
Transportable, local	1	Z*
Not transportable		
Report Writer	8	Y* OR Z* (1st place not numeric 0-9)
Report	4	Y* OR Z* (1st place not numeric 0-9)
Report group	3	Y* OR Z* (1st place not numeric 0-9)
Library	7	Y* OR Z* (1st place not numeric 0-9)
Standard layout		
SAPscript	16	Y* OR Z* (include Development ID)
Form	4	Y* OR Z*
Standard text ID	32	Y* OR Z*
Standard text name	8	Y* OR Z*
Style		
Screen	4	9000 - 9999 If the screen does not belong to a module
Set	12	Y* OR Z* (1st place not numeric 0-9)
SPA/GPA parameter	20	Y* OR Z*

Spool	16	Y* OR Z*
Layout type	8	Y* OR Z*
Font family	8	Y* OR Z*
Device type	8	Y* OR Z*
Page format	8	Y* OR Z*
System barcode		
Standard role	8	9*
Standard task	8	9*
Structures/structure fields	30	Y* OR Z*
SYSLOG message ID	2	Y* OR Z*
Table	10	Y* OR Z* (T9*, P9*, PA9*, PB9*)
Pool and cluster tables	16/30	PS9*, PT9*, HRT9*, HRP9*
Transparent tables	16	HRI9*
Table field		YY* ZZ* (if possible in append) Y* OR Z*
Transaction code	20	Y* OR Z* (also refer to previous sections)
Type (ABAP)	5	Y* OR Z*
View	16/30	Y* OR Z*
Help view	16/30	H Y* H Z*
View cluster	30	Y* OR Z*
View maintenance data	-	Reserved in TRESC
View content	-	Reserved in TRESC
Table content		
Workflow object type	10	Y* OR Z*

3. ABAP/4 STANDARD CONVENTIONS

The purpose of this section is to define attributes and/or naming standards for the following parameters/objects associated with the ABAP/4 programming language:

- Program Object Attributes
- Internal Structures
- Variable Names
- Error Messages
- Memory Storage ID's
- PF Status
- External Files
- Variants.

3.1. Program Attributes

Filename: T:\Project Prep Deliverables In Review\1.1.21_TECH_SD_Development_r1.2.doc

The purpose of this section is to define the allowable attributes for programs created in the SAP environment. The following are the allowable attributes to be set at the creation of a program:

Type:	Executable Program (Report/Conversion/Interface/Utility) Include Module Module Pool Function group Subroutine Pool
Status: Customer Program System program Test Program	Production targeted programs (Customer Program) Production or other Basis utility program Test program not intended for production
Application:	Reference R/3 Business Objects (see matchcode values)
Authorization Groups:	Restricts access/execution to users in this authorization group - if it is deemed necessary to control the access privileges by authorization group.
Logical Database	Choose LDB if desired for report
Selection Screen	Only if LDB is used; choose Selection Screen if multiples exist
Fixed Point Arithmetic:	Always mark this attribute unless working with whole numbers only.

* Unless otherwise specified above the given attribute value should be left blank or not set.

Program Internal Structures

The purpose of this section is to define the naming conventions for internal structures. The following are the internal structures to be discussed:

- Internal Table
- System Table

3.2. Internal Table Naming Convention

Internal tables are defined within programs and used to hold blocks of records. Internal tables will always have a variable prefix of "t_". The following is an example of an internal table variable:

T_CUST

3.3. System Table Addressing Convention

The system table contains several fields used to address system parameters. Reference all system fields by 'SY-xxxxxx'.

3.4. Program Variable Names

The purpose of this section is to define the naming conventions for program variables based on variable types. Program variables are defined using the DATA statement. The PARAMETER statement can also be used to define variables, which are presented to the user as input variables at program execution. Special variables for selection criteria can also be defined using the SELECT-OPTIONS statement. The following are the variable types that are addressed:

- Flags
- Indexes
- Constants
- Work-fields
- Parameters
- Field-symbols
- Ranges.
- Form (Subroutine) Parameters
- Local Variables
- Global Variables.

Underscores ‘_’ are to be used to separate words or letters that define a variable instead of hyphens. Hyphens ‘-’ are used for referencing sub-fields in a record string or data structure. It is recommended that data dictionary fields be used whenever possible for consistency and type checking.

3.4.1. Flags

Flags are variables that contain only one of two values. These variables will be defined as character of length 1. In all cases, the following values must be complied with:

‘ ’ = "NO", "FALSE", or "OFF"

‘X’ = "YES", "TRUE", or "ON"

These variables types are to be suffixed with ‘_FLG’. The following are examples of variable names to be used:

W_EOF_FLG

W_CONTINUE_PROCESSING_FLG

3.4.2. Indexes

Index variables are numeric fields used for indexing internal tables. These variable types are to be named the referring table name suffixed with ‘_IDX’. The following are examples of variable names to be used:

T_CUSTOMER_IDX

T_TEMP_SALES_IDX

3.4.3. Constant Fields

Constant variables do not change during the execution of a program. Whenever possible the literal itself should be used. These field types should be prefixed with 'C_'. The following are examples of variable names to be used:

```
C_BASE_CLIENT LIKE SY-MANDT VALUE '001'.
```

3.4.4. Work Variables

Work variables are for general purpose use. These are any variable types that can not be defined to the above categories. These variables may contain alphanumeric, numeric, packed, binary, date, time, or hexadecimal values.

Variable types of this category must be prefixed with 'W_' and suffixed with one of the allowable suffixes from Appendix C. Suffixes are to be meaningful to the variable being defined. The following are examples of variable names to be used:

```
W_TEMP_SALARY_NB  
W_EMPL_ADDRESS_TX  
W_LAST_UPDATE_DT  
W_LAST_UPDATE_TM  
DATA: BEGIN OF W_TEMP_CUST_DS  
..<Variables / Includes>..  
END OF W_TEMP_CUST_DS
```

3.4.5. Parameter fields

Parameters variable are used to allow data entry at the selection screen before program execution. These variables have a limited length of eight (8) characters. Variable types of this category must be prefixed with 'A_'. The following are examples of variable names to be used:

```
A_CUSTNM  
A_FISCYR
```

3.4.6. SELECT-OPTIONS fields

The SELECT-OPTIONS <Field name> for <Table field> statement generates a program variable used for selection criteria. This allows a user at the selection screen to enter in a range of values. The variable generated is actually a system-generated table containing <variable>-highvalue, <variable>-lowvalue, <variable>-sign, and <variable>-option for the table field selected. Variable types of this category must be prefixed with 'S_' and have a limited length of eight (8) characters. The following are examples of variable names to be used:

S_LIFNR

S_BUKRS

3.4.7. RANGE fields

The RANGES <Field name> for <Table field> statement generates a program variable used for selection criteria much like the SELECT-OPTIONS. The variable generated is actually a system generated table containing <variable>-high value, <variable>-low value, <variable>-sign, and <variable>-option for the table field selected. Variable types of this category must be prefixed with 'R_' and have a limited length of eight (8) characters. The following are examples of variable names to be used:

R_LIFNR

R_BUKRS

3.4.8. Form (Subroutine) Parameter fields

Form (Subroutine) Parameters variable have a limited length of eight (8) characters. Variable types of this category must be prefixed with 'P_'. The following are examples of variable names to be used:

P_PERNR

P_BUKRS

3.4.9. Local Variables

Local variables are declared and used in a subroutine. These Variables are created only in particular form or report. They are relevant to only to that form or report.

L_PERNR

L_BEGDA

L_ENDDA

3.4.10. Global Variables

If a variable is used frequently across application in forms, function modules etc, it should be declared as a Global Variable. (Note: When you change an existing global variable, this may affect a number of reports and forms which already use it)

G_PERNR

G_DOCUMENT_NO

G_ENDDA

3.5. Error Messages

The purpose of this section is to define the naming conventions to be used for standard error messaging. Standard SAP error messages should be used whenever possible to eliminate redundancy. If it is determined that additional error messages are to be generated, assign them according to the guidelines described in this section. The following are the three parameters to be defined in this section:

- Message ID
- Message Severity
- Message Number .

3.5.1. Message ID's

Message ID's are two character names given to a group of related messages. The following are the naming standards by position for message ID's:

1	"Z" Required by SAP
2	Application area code (Ref. Appendix A)

3.5.2. Message Severity

Message severity defines the action performed upon execution of the message. The following are the allowable message severity codes:

A	"Abend" - the transaction aborts and the user is unable to continue.
E	"Error" - entered data is invalid. The user must modify or reenter the data.
W	"Warning" - the program allows the user to override the particular warning and continue, or modify the data, which caused the warning.
I	"Informational" - displays information that may be of interest to the user.
S	"Success" - a success message is an informational message displayed on the next screen. Such as "Transaction completed successfully".

3.5.3. Message Number

Message numbers are the unique 3-character identifier of a message. Message numbers range for 001 to 999. The next available should be used.

3.6. PF-Status

PF Status is a custom status bar defined to be used by a screen or interactive program to enable user actions. The standard SAP format must be used. The table below defines the usage of standard function keys. These function keys do not have to be enabled but when used, must comply with the standards defined in this section. These function keys are not to be replaced by user defined keys.

PF Key	OK-CODE	Description
01	HELP	Display help screen for current field

02	PICK	Select entry (same as double-click).
03	BACK	Exit current transaction. Overrides mandatory entries.
04	LIST	List Possible Entries.
10	MENU	Menu bar.
11	SAVE	Save entries.
12	RW	Cancel current request, Mandatory entries override.
13	PRI	Print.
14	DLT	Delete.
15	EXIT	Exit (Fast EXIT) leave transaction. Same as PF03.
21	PP--	First page of document.
22	P-	Previous page.
23	P+	Next page.
24	PP++	Last page of document.

3.7. External Files

External files are to be defined as logical files via the transaction FILE. This allows for consistent naming of files without having programs being dependent on the actual system file name

3.8. Variants

Variants are program specific objects, which define, set parameter entries required at program execution time. Variant names will consist of 14 characters formatted as follows:

Position

1-10	The associated report program name (Ref. 1.1)
11	Underscore " "
12	"V" for variant.
13-14	2-character numeric, ranging from 01 to 99, starting at 01.

4. ABAP/4 PROGRAMMING STANDARDS AND PRACTICES

The purpose of this section is to define the basic standards that all developed programs must comply with and describes the rules surrounding the use of ABAP features and components.

4.1. GENERAL RULES

- ABAP programs that update master and transactional data **MUST ALWAYS** use SAP transaction codes (where transaction codes are available) by utilizing BDC or 'call transaction' utilities. The only exception to this would be the use of a BAPI or "direct input" program provided by SAP. These methods ensure that logical units of work, rollback, locking operations and edits are performed. SAP-supplied tables **MUST NEVER** be updated directly by custom programs.

Filename: T:\Project Prep Deliverables In Review\1.1.21_Tech_SD_Development_r1.2.doc

- ABAP programs MUST NEVER be used to update configuration tables.
- SAP-delivered ABAP programs, Dynpros, SAP transactions and Batch programs should never be changed without the approval of the Development Team Leader. SAP approval will also generally be required. If it becomes necessary to modify one of these objects, then, if possible, the object should be copied to a new name using SAP naming standards, and modifications should be made to the copied object.
- Before writing any code, make sure that no existing programs or function modules, either custom or SAP-supplied, satisfy the coding requirements.
- If custom coding is necessary, attempt to write the module so that it is re-usable and can be stored in a central library. Function modules are an example of this approach.

4.2. ABAP Program Layout

The goal of structuring an ABAP program is to facilitate post build maintenance. Invariably this will be undertaken by someone other than the original author, it is therefore imperative that a clear consistent approach be taken. This section details the guidelines that should be taken with all ABAP programs and should be used in tandem with the standard ABAP template supplied with these standards.

Always utilise the events START-OF-SELECTION & END-OF-SELECTION even if logical databases are not being used. These events serve to highlight the MAIN part of the program and these should contain the major SELECT groups and display to any programmer the key processes of the program.

The *Pretty printer* formatting command can be used to automatically format code for neatness and readability. The following structure should be followed as a guide to event and section sequencing:

Header Declarations

PROGRAM STATEMENT
COMMENT BLOCK
TABLES
TYPES
CONSTANTS
STRUCTURE DEFINITIONS
INTERNAL TABLE DEFINITIONS
FIELD-GROUPS and FIELD-SYMBOLS
DATA
PARAMETERS and SELECT-OPTIONS

Processing Block

INITIALIZATION
AT SELECTION-SCREEN
START-OF-SELECTION
GET (if using logical databases)

END-OF-SELECTION

Interactive Events

AT LINE-SELECTION
AT PFnn
AT USER-COMMAND

Page Controls

TOP-OF-PAGE
END-OF-PAGE

Subroutines

FORMS

4.3. Program Structure

All subroutines (Forms) must be placed after the END-OF-SELECTION statement.
Comments statements are to be used for the following conditions:

- Before each major section
- Before Small sections of code
- To clarify complex or unclear logic.

Major sections of code are defined as ABAP events, forms, and large code sections that perform a definable task. Small sections of code are defined as 20 or fewer lines of code that perform a definable task.

Dead Code should be removed from the program, i.e., the fields which are never referenced and code which can never logically executed.

4.4. Main Documentation Block

The main documentation block contains a detailed description of the program, maintenance history, and tables referenced. The following contains the list of required entries in the main documentation block:

Program Name and Title.

Date and Author of program.

Purpose of the program. Freeform text, should be descriptive.

List of input/output parameters.

If writing Include module or Subroutines.

List of SAP tables updated.

For each, define: Input, Output, or Both.

List external tables or files accessed by the program.

Filename: T:\Project Prep Deliverables In Review\1.1.21_TECH_SD_Development_r1.2.doc

For each, define: Input, Output, or Both.

List the programs possible return codes, if applicable.

Revision table must contain an entry for each revision including:

Date of change.

Programmer name.

Description of change.

Development request number.

EX:

```
*****
* Author           : Author
* Creation Date    : 12/03/2002
* Specifications By : Author
* Development Request No: DEVK9nnnnn
* Addl documentation :
* Description       : This program contains the form routines for
* processing the BDC structures for BATCH INPUT processing
*
* Inputs           : NONE
* Outputs          : NONE
*
* External Routines : NONE
*
* Includes         : NONE
* Error Handling    : N/A
* Recovery Procedures : N/A
*
* Modifications Log
* Date  Developer   Request ID   Description
*
*****
```

4.5. Statement Formatting

Use Pretty print to automatically indent all statements.

Chaining of like statements is acceptable but not required.

4.6. Include Code

Creating Include modules is recommended for performance and legibility. Use of function modules for common routines is preferred over excessive use of Include modules.

4.7. Messages

All Error messages are to be implemented via MESSAGE ID's. Any text messages must use NUMBERED-TEXT, defined via include text. The purpose of this is to make any output message text language dependent.

4.8. Report Formatting

The reports should be coded with "NO STANDARD PAGE-HEADING" specified in the "REPORT" statement. Then, in the "TOP-OF-PAGE" event, use the function module Z_HEADER for writing project standard header.

Sample:

```
*****
*          T O P - O F - P A G E          *
*****
```

TOP-OF-PAGE.

```
FORMAT RESET.
FORMAT COLOR COL_HEADER INTENSIFIED.
CALL FUNCTION 'Z_HEADER'
  EXPORTING
    FLEX_TEXT = W-SEARCH_CRITERIA.
FORMAT RESET.
WRITE: / T_HMAPPED-OWNER,
       T_HMAPPED-SIZE,
       T_HMAPPED-MONTH,
       T_HMAPPED-DAY,
       T_HMAPPED-TIME,
       T_HMAPPED-DIR_FLAG,
       T_HMAPPED-FILENAME.
ULINE.
FORMAT RESET.
```

For program parameters, use Selection Texts to specify literals that will appear on the screen via editing Text Elements of the program. This will enable the literal to be implemented in multiple languages. Do not use hard-coded literals. Use Text Elements to handle literal text that is printed on a report, the advantage is easier maintainability.

```
WRITE: / TEXT-001. " writes error message
WRITE: / 'Error'(001).
```

There should be no hard-coding of date formats in the programs, The date format of the users' default information (SU50) should be used to output any date.

There will be no hard-coding of currency output formats in our programs. The decimal notation of the users' default information (SU50) should be used to output any currency value.

The European standard paper size is A4 (210 x 297 mm) whereas the US standard is Letter size (8.5 x 11 inches). A4 is a little bit longer and a little bit narrower than Letter size paper. The reports should be designed so that they will work on either paper size, so we don't end up creating two versions of a report simply to accommodate the different dimensions. This means we should design our reports to fit on the minimum dimension of each, in Landscape orientation the report should be designed to fit Letter length (11 inches) and A4 width (210 mm).

Blank lines are to be specified using 'SKIP <n>' as opposed to multiple 'WRITE /' statements. Use NUMBERED-TEXT for all screen text to be displayed. This is defined via include text and is language dependent.

4.9. Batch Data Input

Batch data input sessions should be limited to approx. 250 transactions per BDC group. This number can vary greatly depending on the quality of the data being processed. Large batch ABAP programs are to be avoided due to the single-threaded nature of the batch environment. These programs are to be broken up into smaller batch programs, where appropriate.

4.10. Data Access

Use logical databases when practical for improved efficiency. Selection and efficiency considerations need to be addressed when defining programs. Data accesses should be compared to their equivalent in ABAP SQL.

Use of direct database SQL is forbidden on any SAP defined object. Any statement within EXEC-SQL and END-EXEC circumvents SAP data dictionary checks and will corrupt system integrity. Direct SQL should also be viewed critically for customer created tables within the SAP environment. All tables should be defined through the data dictionary to ensure proper variable and structure type checking.

Data access should be checked whenever performed for invalid return codes. Use the 'CHECK' or 'CASE' statement as opposed to multiple if statements to screen incoming data or return codes.

4.11. Data Update

SAP database files are only to be updated through SAP provided code. This can be through the use of existing SAP transactions or through the use of Batch Data (BDC) driven screen processing.

4.12. External File Usage

Structures of external files are to be defined in the data dictionary as internal tables using fields, data elements, domains, and, where appropriate, check tables.

4.13. Variants

Variants are to be defined as needed to default parameters required for program execution. All batch programs with defined parameters require variants to process. Refer to Variant Naming Standards.

4.14. Computations

There is no performance issue based on the variations presented below for calculations.

COUNT = COUNT + 1 is equivalent to ADD 1 to COUNT.

4.15. Subroutines

Use local variables whenever possible within subroutines for modularization purposes. However, frequently used subroutines within one program call should use global variables to eliminate creation time of those local variables.

It is recommended to pass data to and from subroutines with the 'USING' statement in the PERFORM. This does not apply to field initialization subroutines and global subroutines.

Use of the 'CHANGING' clause is up to programmer preference but is good for subroutine documentation.

Subroutines that are to be called by multiple programs should be created as a Function Module. External subroutines should also be implemented as Function Modules.

4.16. Authorization Checking

Authorization objects should be checked at the selection screen of report programs or at the transaction code level. All check objects created must be documented thoroughly and given to the security administrator. The authorization check should be modularized to have a central access to all authorization objects. An include in the code could give easy access to the already developed security/authorization checks. It is good practice to use Logical database in HR and this will ensure proper authorization checks.

4.17. Lock objects

Lock objects should be created as function modules (Enqueue / Dequeue) in a customer function group.

4.18. Debugging

All debugging code must be removed from a program before migration to a controlled environment (i.e. Quality Assurance, Production). This includes breakpoints and any code associated with testing.

4.19. Standards for Specific Statements

4.19.1. APPEND TAB

Use the SORTED clause only when:

- There are less than 100 rows
- Rows are already in sorted order

If an internal table must be sorted and the above cannot be satisfied, fill the table in unsorted order and then sort specifying key fields. To remove duplicates, if required, collect the rows from the sorted table into an auxiliary table.

4.19.2. ASSIGN

Use of field symbols is discouraged unless necessity dictates. Field symbols, when used, should be documented in program comments when defined and whenever used.

4.19.3. AT PFnn

Use the ' AT USER COMMAND' instead of 'AT Pfn'. This ensures proper response to the user command and is more legible.

4.19.4. CHECK

Use check statements whenever possible instead of nested IF's.

4.19.5. COMPUTE

The COMPUTE statement can be used although typing out the expression using operators and parentheses is more legible.

4.19.6. DATA

Define variables within the DATA statement in accordance with Program Variable Names. Align variable definitions in a readable and consistent format. The LIKE parameter in the DATA statement is required whenever possible. This allows for consistent field definitions between program variables and equivalent SAP variables.

Although ABAP/4 initializes all fields at program execution, it is always good form to initialize your variables using the CLEAR statement. Always clear data structures and variables before repopulating, especially in loops. This highly recommended in function modules since multiple calls to the same function module within one program execution will not refresh the function modules variables.

4.19.7. EDITOR-CALL

Never to be used for table updates since it circumvents any data type checking.

4.19.8. CHECK, EXIT, REJECT, STOP

Use these statements to suspend processing and/or skip remaining unnecessary processing for improved performance.

4.19.9. IF/CASE

Code only one condition per line for legibility. Use parentheses to group conditions.

Use the CASE/WHEN statement structure whenever applicable to avoid nested IF statements of more than 3 levels.

4.19.10.MOVE-CORRESPONDING

This statement should be used with care since only fields with identical names are moved. If moving a data structure to an equivalent structure, move the entire data structure in one statement using <LFA1_DS> = *<LFA1>.

4.19.11.READ TABLE

When reading a table with the 'WITH KEY' option, ensure that the key is filled from the left most field in the table. This is most efficient with a binary-sorted table. Build and read long internal tables with the BINARY SEARCH addition.

4.19.12.SORT

Always specify the sort fields on the SORT statement for legibility and maintainability, including when the system default is used.

4.19.13.COLLECT

The COLLECT statement for internal tables can be very CPU intensive for internal tables containing more than 50 rows. The following are two alternatives to be used for COLLECTING internal tables greater than 50 rows:

Alternative 1.

READ itab WITH KEY new-key BINARY SEARCH.

CASE SY-SUBRC.

WHEN 0. new-amt = new-amt + itab-amt.

MOVE ...

MODIFY itab INDEX SY-INDEX

WHEN OTHERS.

MOVE ...

INSERT itab INDEX SY-INDEX.

ENDCASE.

Alternative 2.

Loop through the already sorted table into an auxiliary table and combine the like records using logical expressions.

4.20. User Interface (GUI)

GUI statuses should be used for interactive report programs and online programs. Use menu bar linking whenever possible to build consistent GUI statuses for screens within a module pool.

4.21. Maintenance/Correction of existing production code

Surround changed code with "BEGIN OF" and "END OF" comment statements with the change request number included.

If major coding changes are to occur, indicate the changes in an extended description and revamp the program. The program, however, must comply with the standards defined in this document at the time the modification is to take place and not rely on the old standards when the program was originally developed. This will ensure the reworked program is up to date on new standards and practices.

4.22. External Data Sets

Define via logical file names using transaction FILE.

See the following example:

```
DATA:      G_PDSNME LIKE FILENAME-FILEEXTERN. "Physical Dataset      Name.

DATA:      BEGIN OF T_TAB OCCURS ...
           END OF T_TAB.

PARAMETERS: P_LDSNME LIKE FILENAME-FILEINTERN "Logical Dataset Name
           DEFAULT 'CUSTOMER_FILE'.

AT SELECTION-SCREEN.
  CALL FUNCTION 'FILE_GET_NAME'
    EXPORTING
      LOGICAL_FILENAME = P_LDSNME
    IMPORTING
      FILE_NAME = G_PDSNAME
    EXCEPTIONS
      FILE_NOT_FOUND = 01.

  IF SY-SUBRC <> 0.
*-----No physical filename can be determined for &
      MESSAGE E016 (MG) WITH P_LDSNME.
  ENDIF.

START-OF-SELECTION.

OPEN DATASET G_PDSNME FOR OUTPUT IN TEXT MODE.

IF SY-SUBRC <> 0.
* Insert error processing here.
ENDIF.
```

```
TRANSFER T_TAB TO G_PDSNME.  
CLOSE DATASET G_PDSNAME.
```

4.23. Internal Tables

The 'OCCURS n' statement should closely match the expected number of rows that the table will contain. An area in the system roll area is reserved based on the table size defined by this statement. If more table rows are appended into the internal table than defined in the occurs clause, records will be stored in either the roll area buffer or paged out of memory. This will cause performance degradation in referencing table entries.

The CLEAR statement should be used to initialize the table header record.

The REFRESH statement should be used to delete all table entries and release any paged out area. It is recommended that before a table is used; perform a REFRESH then a CLEAR in this order. A REFRESH does not clear the header record.

The FREE statement should be used at the end of the program in the wrap-up section. Performing a FREE will release the storage area in the system roll area allotted to the table and deletes all lines.

MOVE used instead of MOVE-CORRESPONDING for efficiency purpose, MOVE-CORRESPONDING is used only when two tables have same field structure.

Specify the sort fields on the sort statement, not leave the fields as default, the more restrictively you specify the sort key, the faster the program will run.

Ex: SORT INT_TAB by K.

LOOP ... WHERE is faster than LOOP/CHECK because LOOP ... WHERE evaluates the specified condition internally.

Ex: Loop at int_tab where K = KVAL.

“...”

Endloop.

Using BINARY SEARCH instead of linear search when internal table has more than 20 entries. If TAB has n entries, linear search runs in $O(n)$ time, whereas binary search takes only $O(\log_2(n))$. And also when reading a single record in an internal table, the READ TABLE WITH KEY is not a direct READ. Therefore, SORT the table and use READ TABLE WITH KEY BINARY SEARCH.

Ex: Read Table Int_tab with Key K = 'X' BINARY SEARCH.

Use the MODIFY variant MODIFY itab ... TRANSPORTING f1 f2 ... for single line, and MODIFY itab ... TRANSPORTING f1 f2 ... WHERE condition for a set of line, to accelerate the updating of internal table. The

longer the table line is, the larger the speed-up is. The effect increases for tables with complex structured line types.

Ex 1. WA-DATE = SY-DATUM.
LOOP AT TAB.
MODIFY TAB FROM WA TRANSPORTING DATE.
ENDLOOP.

Ex 2: TAB-FLAG = 'X'.
MODIFY TAB TRANSPORTING FLAG
WHERE FLAG IS INITIAL

Use DELETE ADJACENT DUPLICATES / DELETE itab FROM ... TO ... / DELETE itab [FROM ...] [TO ...] WHERE ... instead of DELETE, to transfer the task of deleting entries to the kernel.

Ex 1: DELETE ADJACENT DUPLICATES FROM
TAB_DEST COMPARING K.

- * Table TAB_DEST is filled with 1000 entries of 500 bytes
- * each, and lines 450 to 550 are to be deleted

EX 2: DELETE TAB_DEST FROM 450 TO 550.

- * Table TAB_DEST is filled with 1000 entries of 500 bytes
- * each, 250 of which match the WHERE condition

EX 3: DELETE TAB_DEST WHERE K = KVAL.

Internal tables can be copied by move just like any other data object. If the table has a header line then the tables need to be referenced by square brackets.

EX: ITAB1[] = ITAB2[].

5. PROGRAM LOGIC

5.1. STANDARD

A program should test the system return code field (SY-SUBRC) after any statements that could potentially change its value unless the outcome of the statement is not important for subsequent processing. The return code should always be checked after any database table read/update statements.

5.2 GUIDELINES

- CHECK is used instead of IF/ENDIF whenever possible
- When Coding IF, nest true testing conditions, so that the outer conditions are most frequently true.
- CASE statements are clearer for legibility and a little faster than IF-constructions. When testing fields equal to something one can use either the nested IF or the CASE statement. CASE statements are clearer and after about five nested ifs the performance of the CASE is more efficient.
- WHILE is used instead of a DO+EXIT-construction, because WHILE is easier to understand and faster to execute.

Do	While
<pre> I1 = 0. DO. IF C1A NE SPACE. EXIT. ENDIF. ADD 1 TO I1. IF I1 GT 10. C1A = 'X'. ENDIF. ENDDO. </pre>	<pre> I1 = 0. WHILE C1A = SPACE. ADD 1 TO I1. IF I1 GT 10. C1A = 'X'. ENDIF. ENDWHILE. </pre>

- LOOP WHERE is faster than LOOP/CHECK because LOOP WHERE evaluates the specific condition internally.

Loop/Check	Loop/Where
<pre> LOOP AT ITAB. CHECK ITAB-NAME1 = KVAL. ENDLOOP. </pre>	<pre> LOOP AT ITAB WHERE NAME1 = KVAL. ENDLOOP. </pre>

6. DATA ACCESS

6.1 DATA ACCESS – STARNDARD RULES

- In a SELECT statement, only the fields that are needed are selected in the order that they reside on the database, thus network load is considerably less. The number of fields can be restricted in two ways using a field list in the SELECT clause of the statement or by using a view defined in ABAP Dictionary. The usage of view has the advantage of better reusability.

Select *	Select with select list
<pre> SELECT * FROM DD01L WHERE DOMNAME LIKE 'CHAR%' AND AS4LOCAL = 'A'. ENDSELECT. </pre>	<pre> SELECT DOMNAME FROM DD01L INTO DD01L-DOMNAME WHERE DOMNAME LIKE 'CHAR%' AND AS4LOCAL = 'A'. ENDSELECT. </pre>

- SELECT SINGLE is used instead of SELECT-ENDSELECT loop if full primary key is known. Otherwise, use Up to 1 Rows. SELECT SINGLE requires one communication with the database system.

Select ... Endselect	Select Single
SELECT * FROM SCARR WHERE CARRID = 'LH'. ENDSELECT.	SELECT SINGLE * FROM SCARR WHERE CARRID = 'LH'.

- Always specify the conditions in the WHERE-clause instead of checking them with check-statements, The database system can then use an index (if possible) and the network load is considerably less. You should not check the conditions with the CHECK statement because the contents of the whole table must be read from the database files into DBMS cache and transferred over the network. If the conditions are specified in the where clause DBMS reads exactly the needed data.

Select + Check statement	Select with Where condition
SELECT * FROM SBOOK. CHECK: SBOOK-CARRID = 'LH' AND SBOOK-CONNID = '0400'. ENDSELECT.	SELECT * FROM SBOOK WHERE CARRID = 'LH' AND CONNID = '0400'. ENDSELECT.

- Use a select list with aggregate functions instead of checking and computing, when try to find the maximum, minimum, sum and average value or the count of a database column, thus network load is considerably less.

Select ... Where + Check	Select using an aggregate function
C4A = '000'. SELECT * FROM T100 WHERE SPRSL = 'D' AND ARBGB = '00'. CHECK: T100-MSGNR > C4A. C4A = T100-MSGNR. ENDSELECT.	SELECT MAX(MSGNR) FROM T100 INTO C4A WHERE SPRSL = 'D' AND ARBGB = '00'.

6.2. DATA ACCESS – GUIDELINES

- Always select into an internal table, except when the table will be very large (i.e., when the internal tables will be greater than 500,000 records). Use “Up to N Rows” when the number of records needed is known.

Select + Append statement	Select Into Table
REFRESH X006. SELECT * FROM T006 INTO X006. APPEND X006. ENDSELECT.	SELECT * FROM T006 INTO TABLE X006.

- Whenever possible, use array operations instead of single-row operations to modify your database tables. The frequent communication between the application program and database system produces considerable overhead.

Single-line Insert	Array Insert
LOOP AT TAB. INSERT INTO CUSTOMERS VALUES TAB. ENDLOOP.	INSERT CUSTOMERS FROM TABLE TAB.

- Use logical database as appropriate. Logical database implicitly executes authorization checking for data access, but even if you just access one table, you will still need to access all the tables above the current hierarchy level in logical database, which creates data access inefficiency.
- Do not use Nested Select. Use 'Inner Join' and/or 'For all Entries' instead. 'For all Entries' is used over 'Loop at itab / Select / Endloop'

6.3 DATA ACCESS – TIPS

- No complex WHERE clauses, since complex where clauses are poison for the statement optimizer in any database system.
- For all frequently used SELECT statements, try to use an index. You always use an index if you specify (a generic part of) the index fields concatenated with logical ANDs in the Select statement's WHERE clause.

Select without index support	Select with primary index support
SELECT * FROM T100 WHERE ARBGB = '00' AND MSGNR = '999'. ENDSELECT.	SELECT * FROM T002. SELECT * FROM T100 WHERE SPRSL = T002-SPRAS AND ARBGB = '00' AND MSGNR = '999'. ENDSELECT. ENDSELECT.

7 ABAP/4 MODULE POOL STANDARDS AND PRACTICES

The purpose of this section is to define the basic standards that all developed programs must comply with when creating or maintaining Module Pools in the SAP environment. This section describes the rules surrounding the use of ABAP features and components specific to Module Pool program types. Coding in the main program must comply with standards and practices defined for ABAP/4.

7.1 Program Structure

The ABAP Programmers Workbench (SE80) should be used to create and maintain all Module Pool programs. Sample flow logic and associated modules for all screens in a module pool is defined in Appendix E.

7.2 Screen Definitions

Screens are to use data dictionary fields for all variables. This does not include multiple selections like radio buttons.

Screens are to be numbered with an entry screen of 100. Reuse screens whenever possible.

7.3 GUI interfaces

GUI interfaces must follow SAP standards. User defined menus are to contain all actions that are applicable to the transaction. Items that are not effective during a screen process are to be disabled. Commonly used selections are to appear as push buttons below the menu bar.

7.4 POP-UP windows

Pop-up windows are allowed and to be used at programmer or customer preference. Programmatic circumvention's of pop-up windows must be included for batch data processing considerations.

8.5 OK_CODE (SY-UCOMM)

An OK_CODE variable must be specified in all screens. This variable is defined via the field list of each screen created (Exists as the last field in the field list).

8 REFERENCES

- 1) BC SAP Style Guide: SAP America, Appendix E; Document No. 5000-7453

Are there any standards or recommendations on selection screens? Use Logical Database to get HR Standard Selection Screen. Using logical database also ensures proper authorization checks. Any standards to be used with reports, e.g. always use ALV for report layout etc.? Can use standard reporting or ALV based on the requirement.

Any words on general advise for unit testing (since this is usually done by the developer first)?

9 UNIT TESTING:

Unit tests are carried out by developers and these tests are essential precondition for quality software, but unit tests must meet certain conditions. One of them is that one test function should test only one thing. Another condition is that tests should not depend on each other. One of the most important conditions is that the tests must be fast and they should test the code in isolation. During the Unit Test make sure that configuration is complete. Identify all the program bugs and configurations issues and fix them in the Development box.

10 APPENDIX A

10.1 Table of Business Object Codes

Business Object	Description
AA	Asset Accounting
AP	Accounts Payable
AR	Accounts Receivable
BA	Basis
BW	Business Warehouse
CA	Cross Application
CO	Controlling
CR	Customer Relationship Management
FM	Funds Management
GL	General Ledger Accounting
GM	Grants Management
HR	Human Resource
MM	Materials Management
PM	Plant Maintenance
PS	Project System
PY	Payroll
SEM	Strategic Enterprise management
SRM	Supply Relationship Management
UT	Utility Billing
YY	Miscellaneous

11 APPENDIX B

11.1 Suffix Codes for Field Names

The following is the list of suffix codes to use when describing variable or field names and elements:

DT	Date
TM	Time
NB	Numeric
PD	Packed Decimal
FP	Floating Point
TX	Text
NM	Name
HX	Hexadecimal

BN	Binary
DS	Data Structure/ String
IX	Index
FG	Flag

12 APPENDIX C

Refer to SAP OSS note #16466 (Customer name range for SAP objects). This note is current as of October 9, 2002.

13 APPENDIX D

13.1 ABAP Check list

What to Check	How to Do It	Completed by
Extended program check <ul style="list-style-type: none"> - Unused code. - Select single without unique key. - Program interfaces - More checks than regular program checks. 	Menu path: Tools – ABAP Workbench – Development - ABAP editor - Program – check – extended program check.	
SQL Trace Reasonable number of records read from each table?	Menu path: System – Utilities – Performance Trace. Trans code: ST05 Trace On. Run program with only obligatory selections. Trace Off. List Trace. To get a summary trace: Trace – Summary – Compress. Review select statements for tables with large numbers of records read.	
Best Performance - Tables Index available? Buffered? How many rows in production? Start selection with smallest table? Views are inner joins. Is this ok?	Menu path: Tools – ABAP Workbench – Development – Data Dictionary. Trans Code: SE11. Enter table name. Click on display. Note file's key. Click on indexes to review available indexes. Click on technical settings for buffering.	
Best Performance in select statements/Internal tables <ul style="list-style-type: none"> - List fields for select statements - Binary searches where possible - Internal tables sorted for future binary and other searches. 	Review program. For coding examples and comparisons: Menu path: System – Utilities – Runtime analysis – Execute. Click on Tips & Tricks. Trans code: SE30. Click on Tips & Tricks	

<ul style="list-style-type: none"> - Free memory if possible (Free itab_mara) - "Select for all entries" is left outer join - Select single or up to 1 rows - Check for empty table before selecting 		
Authority Check Authorization group (program attribute) Transaction code auth check Any other necessary?	Auth Group: From ABAP editor use menu path: Goto – Attributes. Search program for AUTHORITY-CHECK	
Menu Attach a transaction code? Attach to a user menu?	Transaction code: Menu path: Tools – ABAP Workbench – Development – Other tools – Transactions. Trans code: SE93. Menu – Area Menu: Menu path: Tools – ABAP Workbench – Development – Other tools – Area Menus. Trans code: SE43.	
Future Flexibility Multilanguage Text International Currency Quantities and Amounts	Multilanguage Text <ul style="list-style-type: none"> - Use text symbols and messages rather than words - If language is part of key to table make sure it's in where clause (spras = sy-langu). International Currency <ul style="list-style-type: none"> - Use currency field Quantities and Amounts Use unit of measure field	
Background and job scheduling usability At selection screen and initialization events are only executed if selection screen is displayed.	Review program. Search for initialization and at selection-screen events.	
Documentation & standards Naming conventions followed. Each form is explained. Nothing hard coded (or if approved declare as a constant).	Review program.	